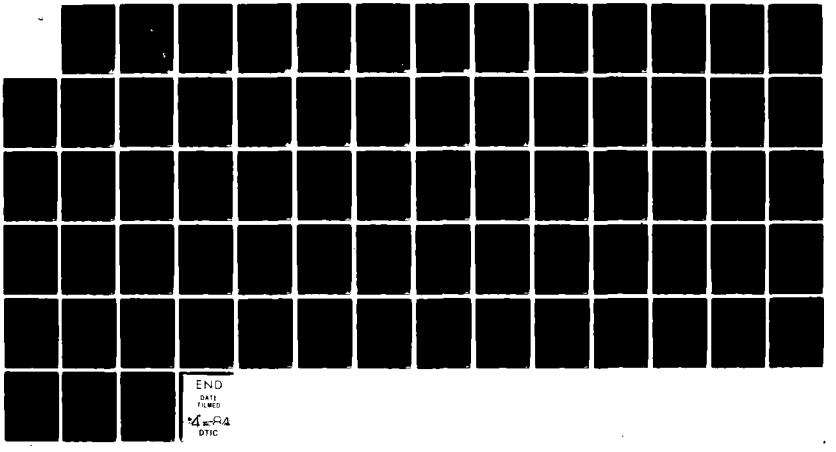
AD-A139 068 A PRODUCTIVITY ENHANCEMENT STUDY OF THE FMSO (FLEET MATERIAL SUPPORT OFFICE) SOFTWARE EFFORT(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA D C BOGER ET AL. 1/1

UNCLASSIFIED NOV 83 NPS-54-83-013 F/G 9/2 NL

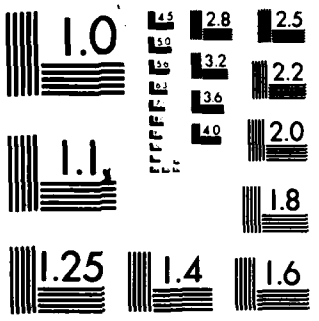


END

DATE FILMED

4-84

DTIC



NPS-54-83-013

NAVAL POSTGRADUATE SCHOOL

Monterey, California

A PRODUCTIVITY ENHANCEMENT STUDY

OF THE FMSO SOFTWARE EFFORT

by

Dan C. Boger

Norman R. Lyons

November 1983

Approved for public release; distribution unlimited.

Prepared for:
Fleet Material Support Office
Mechanicsburg, PA 17055

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Commodore R. H. Shumaker
Superintendent

David A. Schrady
Provost

The work reported herein was supported by the Fleet Material Support Office, Mechanicsburg, Pennsylvania.

Prepared by:

DAN C. BOGER, Assist Prof
Dept of Administrative Sciences

NORMAN R. LYONS, Assoc Prof
Dept of Administrative Sciences

Reviewed by:

RICHARD S. ELSTER, Chairman
Dept of Administrative Sciences

Released by:

KNEALE T. MARSHALL
Dean of Information and Policy Sciences

A PRODUCTIVITY ENHANCEMENT STUDY OF THE FMSO SOFTWARE EFFORT

by

Dan C. Boger
Norman R. Lyons

Department of Administrative Sciences

Naval Postgraduate School

Monterey, California 93943

November 1983

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NPS-54-83-013	2. GOVT ACCESSION NO. AD-A139 068	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Productivity Enhancement Study of the FMSO Software Effort		5. TYPE OF REPORT & PERIOD COVERED Technical Report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Dan C. Boger and Norman R. Lyons		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS N0036782POM4670
11. CONTROLLING OFFICE NAME AND ADDRESS Fleet Material Support Office Mechanicsburg, PA 17055		12. REPORT DATE November 1983 13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/ DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Productivity enhancement, Development tools system, Software development, Project management

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report presents the results of a productivity enhancement study on the software development activities of the Fleet Material Support Office, Mechanicsburg, Pennsylvania.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

SUMMARY

This is the report of the productivity enhancement study of the FMSO software development effort. This study is an intial effort to identify candidate projects for productivity improvements. We do not attempt a detailed analysis of FMSO problems. Instead, we try to adopt an overview of the organization and its problems as they appear to outsiders. It is the opinion of the authors that FMSO is well managed and that employee morale is generally good, but that the organization faces serious challenges in both the near term and the long term. Substantial changes will have to be made in the way the organization does business to keep FMSO viable in the future.

The major recommendations in this report are:

1. FMSC should begin work on a Development Tools System that will support computer programming work, documentation and software management. This should be a unified system (all parts of it can communicate with other parts) but not necessarily a single computer system.
2. The physical facilities at FMSO are below the recognized standards for supporting a software development operation and should be upgraded.

3. Some areas of software management need to be improved. ~~Notably,~~ A better project planning and tracking system needs to be put in place. Generally, FMSO's software management effort is well directed.

The points covered in this report are:

1. An overall view of the FMSO systems effort.
2. A discussion of the type of productivity enhancing effort that should be made.
3. A proposal for the installation of a Development Tools System at FMSC.
4. An outline of the facilities improvements that should be made to improve productivity and encourage continued high employee morale.
5. Project estimating techniques and their usefulness.

CONTENTS

Chapter 1

BACKGROUND

1.1 INTRODUCTION TO FMSO

The Fleet Material Support Office is an unusual Navy command. It handles a variety of responsibilities for the Supply Corps, but most of its work is to function as a Central Design Activity for various supply and financial computer systems. In effect, FMSO is the information systems arm of NAVSUPSYSCOM. The major mission areas for FMSO are:

1. Central Design Agency activities.
2. Management of the Navy's Retail Stock Fund.
3. Operations analysis activities.
4. Supply operations support.
5. International logistics.

The CDA activity consumes 80% of FMSO's resources, and it is the area that we will be concerned with in this report. The major concern of this study is to focus on ways to increase the productivity of the CDA activity.

The major functions supported by the CDA activity are:

1. Uniform Automated Data Processing Systems (UDAPS).
   a) Uniform ADP System for Inventory Control Points (UICP).
   b) UADPS Stock Points (UADPS-SP).

c) Level II/III Stock Points.

d) Disk Oriented Supply System (DOSS).

2. Headquarters Financial Systems.

3. Management Information System for International Logistics - MISIL.

4. Special Data Processing Systems Projects.

a) RMM&E - Requisition Material Monitoring and Expediting.

b) Trident logistics data.

c) NALCOMIS.

d) NAVADS - Navy Automated Transportation Data System.

e) NAVADS - Navy Automated Transportation Documentation System.

f) Resolicitation.

g) SPAR.

The list of responsibilities placed on FMSO is impressive. If it were a private organization, it would be a major software house or computer company. With approximately 1,360 employees FMSO has a staff that is about 200 smaller than Apple Computer. The employees charged with the CDA activity have to maintain a library of computer systems consisting of approximately 10,000 - 12,000 programs totaling on the order of 20 million lines of code. In industrial terms, this system library is about what one would expect to find in a major high technology company that employed around 200,000 people. This library has been in development for 10 to 20

years. Again, if industrial yardsticks apply, then it is to be expected that FMSO has spent between one and two billion dollars in developing this code.

1.2 GENERAL PROBLEMS AT FMSO

Some of the problems that beset FMSO would occur in any information systems group in any organization. Information systems activity is generally a service ar· This means that others in the organization do not rea y appreciate the problems involved in developing software have little idea about effective ways of doing it. Th expect the service to be available when they want it and in the form that they want it. The result is that an information systems group can develop serious problems in its relations with its upper level management and its customers. The group has little control over planning or resource allocation for its area, but it tends to get blamed for everything that goes wrong. In FMSO's case, this problem is made worse by the fact that their superiors are in Washington, and their major customers are spread all over the globe. The reputation of the organization suffers as a consequence even when its problems are not of its own making.

Besides these general sorts of information systems group problems, there is another set of problems that arises for computing groups working for the government in general and for the Navy in particular. During the fifties and sixties,

computers were generally recognized as useful, but they were very expensive and difficult to manage. As a result, a whole set of regulations grew up around the use and procurement of computers with the Brooks Bill being the item most often cited. The effect has been to require lengthy justification for any computer procurement.

The ironic thing is that computers have gotten much cheaper since these regulations were first put into effect. Computer power that would have required a mainframe twenty years ago can now be purchased at K-Mart in the toy department. For computer systems costing between $10,000 and $100,000, the justification of the system is one of the most expensive accessories on the machine. There have been two side effects of this. The first is to make managers reluctant to procure equipment even though it may be of considerable benefit to the organization. For the individual manager, a procurment effort means that his staff is occupied with the paperwork required to purchase a computer instead of being able to do their regular jobs.

Another problem that affects FMSO is the Navy attitude toward shore facilities. There seems to be an unwritten policy in the Navy that the first priority should be given to the fleet while shore and support facilities are of secondary importance. The socialization of Naval officers also leads them to accept shore facilities that are less than ideal. Shipboard life involves a lot of crowding and dis-

comfort. No matter how bad a shore facility may be, it is likely to be more comfortable and spacious than a shipboard facility. Unfortunately for FMSO, the standard of comparison is not shipboard software development facilities (if there were such a thing). It is the numerous software development facilities springing up all around the Harrisburg area. It will be irresistable for FMSO employees to compare their working conditions with those available at places like EDS and CACI.

These comments apply to both the physical plant at FMSO and to the computer systems upon which development is done. The computers for which FMSO does development work must be among the oldest currently operating. This is a costly proposition from many points of view. For the individual programmer, it is costly because he falls behind technologically. Computer personnel are an unusual breed. Of all the professions, they hold professional development in highest regard. This is natural considering that computer technology changes rather quickly, and that any individual who falls behind is likely to find himself out of a job. FMSO has done a good job in making professional development training available to its staff. This is probably a major reason for the remarkable loyalty to the organization we observed there.

There are other problems in trying to deal with older technologies than just personnel considerations. Both the

equipment and design philosophies for operating systems have changed considerably since FMSO's equipment was installed. Magnetic tape oriented systems for data processing are now a thing of the past. Magnetic tape is cheaper than disk, but its use requires a great deal of operator intervention. There are too many chances for error in the use of tape. In a disk oriented system, the process of calling files and setting up jobs is done automatically without human intervention. The disk system may be more costly to install, but the elimination of tape handling errors makes it a good deal cheaper in the long run.

The same is true of older operating systems. Such systems generally called for more operator intervention. This opened up more chance of error. The newer philosophies in operating systems call for "programming the idiot out of the loop" - that is, designing the system so that it rarely calls on humans for decisions. A final point in the operation of older systems is the maintenance problem. As a system ages, the manufacturer of the system becomes less interested in performing software enhancements and updates. He naturally wants to concentrate on newer products, and eventually the software on the older system becomes obsolete. If the customer does not upgrade his hardware, he gets left behind in the evolutionary process of system development.

1.3 PROBLEMS IN REACHING A SOLUTION

The problems mentioned above combine to make long term solutions very difficult in this environment. Acquisition of new computer systems or the institution of long range changes can take years in the Navy environment. An example of this is the ICP Resolicitation project. This has been underway for about eight years now, and the first machine should come on line in 1984 (if all goes well). This is an unconscionably long time for a systems change. In an industrial enviroment, this should take no more than a year with only a few months spent on the study portion.

A critic of FMSO might argue that this only proves that the machines were unecessary in the first place and that the government has saved itself eight years of computer expense by staying with the old equipment. This is all quite true. The machines are "unnecessary" in the sense that there is always another way to do a computer job. In this case, the computer savings were generated at the expense of personnel costs, project delays and degraded service for the naval supply system.

Unfortunately, the personnel costs do not get charged off to specific information processing systems in quite the same way as a computer. If they don't appear on anybody's bottom line, then there is a tendency to regard these costs as not being real.

The Univac 495's that NAVSUP uses to manage the ICP's were obsolescent when the system was installed in 1965. By now, they are hopelessly out of date. What this means is that because of the Navy's "can do" attitude, these machines have been kept going long past their useful life. The price for this is more operations personnel, time spent on systems development and changes to the operating system, and time spent fine tuning FMSO's applications to get the most possible "bang per buck" out of a Univac 494. There is little doubt that ICP's Univac 494's have been tuned so that they are operating more efficiently (where efficiency is measured in computer time only) than any Univac 494's in history. Why anyone would want to do such a thing is another question.

1.4 CONCLUSIONS

The Commanding Officer of FMSO faces several significant challenges. The organization has some real long range problems. These include systems upgrades, improvements that need to be made to take advantages of new technologies and improvement of the physical environment. The steps we recommend to solve some of these problems are going to generate short range chaos. A CO's tour of duty is only two years. If a new CO came in and accepted all of our recommendations on the first day of his tour, then by the end of his tour FMSO would be in a much more disrupted state than

when he took over. In fact, the same would probably be true of his successor, and it would not be until the third CO in the sequence that we would begin to see payoffs from some of the productivity measures we will recommend (about five years out). FMSO is already engaged in a number of steps to solve the problem areas we observed. Things seemed to be moving in positive directions and the management was well aware of the challenges facing before this report was written.

Chapter 2

APPROACHES IN IMPROVING PRODUCTIVITY

2.1 INTRODUCTION

There are many possible approachs to improving productivity in software development. There is no one magic technique that will guarantee results under all conditions. The most effective techniques to apply in improving productivity will depend on the current status of the organization, its level of expertise, and the type of systems and management environment in which it operates. In addition, the approaches to productivity improvement given in the literature tend to be interdependent. They cannot be applied separately or piecemeal and have any chance of achieving success. For example, modern software management techniques cannot be used effectively in an antique computing enviroment. Most of the productivity improvement techniques are highly dependent upon interactive computing environments, sophisticated development tools and the ability to transfer both development code and administrative data quickly among the individuals involved in a project. On the other hand, high technology by itself is no guarantee of a productive environment. A productivity improvement program needs a well thought out managment plan combined with the latest

technology. In this document, we will try to lay out the aspects of a productivity enhancement program for FMSO.

2.2 PROBLEMS IN DEFINING PRODUCTIVITY

The first problem with productivity in a software environment is deciding what it is. This may sound odd at first because everyone thinks that they know what productivity is. In a manufacturing environment such as the automobile industry, it is not too hard to come up with definitions for productivity. An automobile is a tangible item. It either works, or it does not. It is built from components that are easy to cost out and a cost for its production can be computed fairly readily.

In software development, this is not the case. It is hard to come to some sort of solid analysis as to just what is being produced by programmers. In one sense, it is not too different from the case of an automobile. Programmers produce programs, and these programs either work, or they do not. But each programmer working on a system produces only a piece of it, and there is no set standard for measuring what these pieces are. The measure most commonly used in industry is lines of code. All we have to do is count the lines of code produced by a programmer, and divide that into the cost of supporting the programmer, and we have a productivity figure that we can use.

But when one begins to examine both the published literature and the possibilities available in the definition of lines of code, one's confidence in this measure begins to slip away. For instance, what do you count? Is every comment line in a program counted, or do we only count executable code? Do we count the lines in a program that contain commands to the operating system, or do we only worry about the source language code? What do we do about code that has been produced for another system and has been re-used for the system that we are trying to analyze? Can we count code that has been produced by a program generator? The list of possible questions is almost endless.

One reply to this is that it does not really make too much difference. All we have to do is choose some reasonable measure and stick with it. Indeed, this is what most organizations do. But, this does make it hard to compare productivity across organizations. It is not uncommon to find "productivity" differences that are almost an order of magnitude apart in comparing two different software organizations[1]. In many cases, much of this difference in productivity comes from differences in the counting conventions that are applied to computer code. It would be a mistake to accept these differences at face value as true differences in productivity.

---

[1] See Barry Boehm, Software Engineering Economics, p. 86 for a table listing the different effort models and a brief comparison of the number of man-months predicted by each for a software development project.

There are other possibilities for measuring the output of a software effort. It is possible to define the basic functions performed in a program, and then count productivity as number of functions implemented[2]. Another possibility would be to count number of programs released. Both of these are somewhat grosser measures than lines of code, and in their own way, they can be as hard to implement. No matter what measure is chosen for productivity in an organization, care should be taken in its application. One does not want to come up with a measure that encourages behavior that is counterproductive. For example, if one chooses lines of code as a measure then checks should be made from time to time to make sure that this is not encouraging programmers to use coding techniques that maximize lines of code. Similarly, if one chose programs released as a measure, then it would be to a programming team's advantage to only try to work on short, simple systems. This would boost their "productivity" measure as defined by the organization. Whatever measure is chosen, it should be applied with common sense, examined frequently and compared with other measures of productivity. Slavish adherence to an inappropriate productivity measure could do more damage to real productivity than not paying any attention to productivity at all.

---

[2] For an example of this approach, see the article by A. J. Albrecht, "Measuring Application Development Productivity".

2.3 PRIORITIES IN IMPROVING PRODUCTIVITY

In improving programmer productivity at FMSO, we have a few problems because there is no currently accepted definition of productivity in place at FMSO. This means that we could institute programs for productivity improvement, but we have no way of measuring how well these programs perform. One approach to this problem would be to set up some productivity measures, gather data and use this data as a benchmark for future productivity enhancement measures. We feel that this would be a bad approach. The problems that FMSO has are serious enough, and the organization is enough behind the current state of the art in information systems technology that we feel that certain measures must be taken without delay. In setting up a productivity improvement program at FMSC, we feel that the following areas should be considered (in order of priority):

1. Automation of the systems development process.
2. Improvement of the physical environment at FMSO.
3. Development of a system of productivity measurement and data gathering to support this system.
4. Development of a system for project planning and tracking based on the productivity measures developed.
5. Continue the work on a set of automated development tools in support of the systems development effort.

All of these problems are serious, and to a certain extent, they must all be attacked simultaneously. We feel, however, that the automation of the software development process is the most important issue to be solved in the near term. The highest priority should be given to the acquisition of a development tools system to aid in both software development and project management. A major problem at FMSO is that development takes place on "test bed" machines. Such machines are set up to meet the needs of the organization for which the software is being developed. They do not have the full set of software aids that one would expect on a modern software development facility (sophisticated text editors, interactive compilers, file transfer protocols, message handling facilities, and word processing text formatters). These tools are not necessary for the ultimate missions of these test bed machines. However, these automated tools are very effective for software development and project management. Further, because the development machines are identical to the actual production machines, the temptation to pre-empt development activity if one of the production machines is down is very strong. The immediate needs of users naturally have a higher priority than long term development projects, and this can result in slowdowns in development. The "test-bed" machines are actually the production machines of the development groups, but users tend to overlook this. It must be recognized that software

development is a highly specialized activity, and that it needs its own production facility. There is no reason why the development system has to be the same machine as the one for which the software is being developed. In fact, developing the software on a different machine could actually serve to make the code more readily portable.

2.4 THE CASE FOR A DEVELOPMENT TOOLS SYSTEM

The development tools system should serve both management and programmers. It is hard to separate the needs of the programmers and the project managers in a large software effort. If anything, the larger the software effort, the more time and effort will be spent in management and documentation issues. For a large system, actual coding is likely to consume about 30% of the effort. The remaining effort is spent in documentation, management and coordination of the diverse elements making up the system. Any development tools system implemented should be able to support these needs. A unified development tools system should be able to support these functional areas:

1. Development documentation.
2. Project management and tracking.
3. Budgeting.
4. Program coding.
5. Program testing.
6. Database development and program test data.

7. Quality assurance.

To those used to older generations of equipment, this may sound like an impossible list of tasks for a single computer. In fact, this list is the rule rather than the exception on modern large scale computers. Most mainframes offer a wide variety of tools that will support this type of environment.

The processors needed to support systems development include:

1. Interactive compilers.
2. Languages with string processing capabilities.
3. Text processing packages for formatting documentation.
4. Sophisticated file handling capabilities.
5. Screen oriented text editors for manipulating both code and text.
6. The ability to transfer data from one user to another quickly and conveniently. This would be used for both automated office type applications and programmer's work bench.
7. Packages for doing statistical analysis.
8. A graphics system for producing documentation and management reports.
9. Automatic typeset facilities for producing "clean" documentation and reducing printing costs.

The idea is that a development system should support a wide variety of both computer related and management related activities. Some may object to the proposal that text processing abilities should be included on the development system. The counter argument would run - "We already have word processing facilities. Why buy more?". What we are proposing here is somewhat different from the normal word processing systems. The development system would be used to tie together a number of different applications, and word processing would be one of them. It is very convenient and cost effective to be able to do both programming and word processing on the same machine. For one thing, it allows you to take the output of a program, reformat it and use it as part of the text in a manual. This is difficult to do on an ordinary word processing system because it involves reentry of data that was already generated by the computer anyway. The strategies for building a development system are discussed in Chapter 3.

2.5 IMPROVING THE PHYSICAL PLANT

Almost as important as the acquisition of a development tools system is improvement of the programming environment at FMSO. The present facilities are inadquate for any type of clerical work, particularly computer programming. FMSO is housed in an old warehouse building. The space inside the building is broken up using shoulder height partitions.

These partitions are of the old-fashioned sort that do not provide much sound insulation. The building itself is noisy and poorly air conditioned. This is the worst environment for software production that we have ever seen.

The present facilities provide about 50 square feet of floor space for each employee. In the computer industry, 100 square feet is considered normal[3]. Programming is an activity that requires somewhat different types of office spaces than a other clerical jobs. The by-products of computer programming (listings, sheets of graphics output, manual libraries) take up a great deal of work space and storage space. To use them effectively requires specially designed work areas and storage areas. A programmer needs a desk for normal work, a work table where he can spread out listings or notes for work (even with a more modern system that de-emphasizes hardcopy, it will be a while before all programmers accustom themselves to this), a terminal for interaction with the computer and storage areas for listings and manuals. It goes without saying that this all should be in the context of a physically comfortable environment. The air conditioning should be adequate to the climate, and the sound insulation should be good. In addition, there have to be conference facilities readily available for meetings of

---

[3] See the description of the IBM Santa Teresa labs which were specifically designed to support software development. The reference is G. M. McCue, "IBM's Santa Teresa Laboratory - Architectural Design for Software Development".

project teams. Program development for a large system is a relatively social activity, and this meeting space is needed for the job as well. The command seems to be well aware of the space problems and is taking steps to remedy them. Some of the problems will be alleviated in the future as systems work moves away from cards and the floor space taken by card files can be reclaimed.

The issue of programming environment is an important one and is addressed in Chapter 4. There are no specific studies relating programming environments to productivity. As we discussed earlier, productivity measures are rather rubbery, and it would be statistically difficult to relate specific productivity measures to all of the possible variables in environmental design. Still, if your building layout is substantially at variance with what is considered normal in the the industry, then your programmers are likely to notice. FMSO is ringed by software houses (EDS, CACI and others), and the program development environments there are likely to come a lot closer to industry norms than they do at FMSO. Eventually FMSO is going to lose many of its best employees to these organizations because they perceive a better environment there. It is a tribute to the management practices in place at FMSO that the employee morale is as good as it is.

2.6 CONCLUSIONS

The observations we have made in this chapter should come as no surprise. From our conversations with FMSO staff, these are well known problems. They do not seem to be widely known outside the organization, however. We feel that FMSO is at a crucial point in its existence. It has to either improve its technical and physical facilities, or it will cease being a viable software development organization. The options are either to improve the facility or to abandon it.

The process of productivity improvement must be attacked on several fronts. The most important is the improvement of the technical and physcial environment. It does not make sense to try to implement modern software management techniques in a horse and buggy technical environment. The improved software management techniques will be helpful in any environment, but they will not be as effective on a non-interactive, antique computer system. Along with the improvement of the system, an acceptable measure of productivity must be developed and installed. All of these changes are evolutionary, and will take a good deal of time. There are no generally accepted productivity measurement techniques in industry. The models used tend to be tailor made for each organization, and the same will be true for FMSO.

Chapter 3

REQUIREMENTS FOR A DEVELOPMENT TOOLS SYSTEM

3.1 INTRODUCTION

Since the Development Tools System is to be installed in a Navy environment, the first thing to consider is the procurement of this type of equipment. To somebody accustomed to industry standards for software, the lead times for government software procurements seem excruciatingly long. The ICP Resolicitation effort at FMSO has been going on for eight years now, and the first machine has yet to arrive. In industry, eight years would be the complete life cycle for the system from the first feasibility study to the final departure of the system at the end of its life. In the past, such long life cycles have guaranteed that the equipment will be obsolete when installed.

Part of the problem is that a great deal of economic justification is required for a government procurement. There is nothing wrong with this per se, but this economic justificatation is always tied to a specific shopping list of hardware, and the whole thing has to go through many levels of approval. Meanwhile, the computer industry changes at a rapid pace, and the list of hardware quickly becomes obsolete. It would be worthwhile to consider the approach taken

by the SPLICE project in acquiring a system. The focus should be on the functions that they system is to serve and not on the specific list of hardware to accomplish those functions. The system itself should be modular and expandable. Once a procurement authorization is in place, it can be used as a vehicle for future upgrades to the system. This is being done in current Navy procurement (including the ICP Resolicitation), and it may help to relieve some of the past problems. The procurement document is regarded as a vehicle for future upgrades and the need for re-justification of upgrades should be avoided. The same sort of focus should be used in the acquisition of a Development Tools System.

3.2 FUNCTIONAL CAPABILITIES REQUIRED

The functional capabilities in the Development Tools System should include:

1. Interactive compilers and debugging tools for system development in the major languages used at FMSO. This would be COBOL and perhaps FORTRAN.
2. Interactive languages with good string processing capabilities for use as tool generating languages. These would be used to generate data bases, and develop automated tools for analyzing computer code (structure and standards checkers would be two examples). Good candidate languages would be PL/I or Pascal.

3. High speed terminals with full screen editing capability. Ideally, there should be a terminal for each programmer.
4. An electronic mail system so that programs, documentation and data could be routed readily among the members of the development team. Such a system could serve as the foundation of the development and management work on FMSC systems.
5. A variety of management tool aids such as statistical packages (SPSS, Minitab), management packages (PERT) and report generating packages (RAMIS II or FOCUS) to aid in controlling the development of FMSO projects.
6. A sophisticated word processing capability that would include the ability to format large documents (the requirements tend to be different than for small word processing systems). Examples of systems of this type would be Script or ATMS.
7. A sophisticated graphics capability for producing both documentation and management reports.
8. An automatic typesetting system tied in with a high speed laser printer.

This is a formidable set of requirements for a single machine. A number of manufacturers supply equipment that could handle these requirements, but it would probably be more economic to consider a Local Area Network (LAN) rather than to try to satisfy all of this on a single machine. The

networking system chosen would be the vehicle for future system growth. Increments to this computer power could be added as needed.

At the base of this network, there would have to be one or more reasonably powerful mainframe systems. These would be required for implementing programming tools and for doing interactive testing of code. Individual editing and word processing functions could probably be handled on smaller "smart" terminals. It makes more sense to download processing onto cheaper micros and minis rather than trying to find a large CPU to handle everything.

3.3 COST JUSTIFICATION FOR THE DEVELOPMENT TOOLS SYSTEM

It will be difficult to do traditional economic analysis on such a system for cost justification. The normal government approach to economic analysis on systems is to determine requirements, cost out a set of alternatives for meeting those requirements and then choose the most cost effective alternative for the system. There is nothing wrong with this, but it does make one rather large assumption at the start - namely that you can determine your "requirements". In FMSO's case, this will simply not be true. The software development environment there is so far behind current technology that the programmers could not even state exactly how they will use the new system. Our experience at NPS is probably instructive. Before we acquired our IBM

3033AP system, we went through the standard justification and benchmarking based on the best guesses we could come up with on how users would use the new machine. These guesses proved to be totally inadequate because our users were very quick to come up with unanticipated uses of the machine.

This is the problem of trying to assess "unmet demand" in advance. In FMSO's case, the problem is likely to be an order of magnitude worse because the systems in place there are so old. It will take the programmers at least a year before they begin to feel comfortable with the machine. Once they do feel comfortable with it, they will begin to use it in ways that are hard to anticipate. A LAN type technology will at least allow you to expand your resources in a modular fashion.

3.4 CONCLUSIONS

FMSO should try to remain as flexible as possible in its acquisition of a Development Tools System. Fortunately, this is relatively easy to do with newer computer systems. It is possible to buy a system as a set of building blocks and integrate and expand the system over time by adding new pieces. FMSO and the Navy generally need to change the way they think about computer systems. A computer system is not a solution to a problem. This type of thinking leads planners to believe that a particular system is good now and forever. A computer system is a part of a problem solving

process. As the process changes, it is probably a good idea to consider changing the system as well. NAVSUPSYSCOM has failed to do this with its computer systems, and it will be a difficult, expensive process to upgrade.

Newer trends in Navy procurement make it easier to acquire useful computer systems by focusing on the functions provided by the system rather than on a shopping list of computer hardware. FMSO needs to look at developing an integrated system that will support program development, documentation, tools development and management. The system developed should be expandable so that the system can grow as FMSO's needs grow. The most promising candidate for this type of system is the local area network approach. The LAN technology is still fairly new, and it may be a few years before there are any clear leaders in this field. In the meantime, FMSO should begin working on an overall development plan and begin acquiring equipment that could provide short term aid and also be rationally fitted into a LAN development in the future.

Chapter 4

PROGRAMMING ENVIRONMENTS AND PRODUCTIVITY

4.1 INTRODUCTION

The concept of programming environments is one that is just beginning to get attention from researchers in productivity. The development of computer software is still in a cottage industry phase. It is not unusual for a programmer to have to write the code, keypunch it (or do the data entry on a terminal), document it, keep track of modules, integrate modules, test the modules, assemble the release package and a number of other chores associated with the project. This would be similar to having an automotive worker be the designer of a car, write the owner's manual, assemble the car and be responsible for doing maintenance work on it. This would require personnel who were much more skilled than current automotive service personnel, and cars would be a great deal more expensive as a consequence.

This problem that programming work has is just an example of general problems in the clerical area. Capital expenditure per worker is lower for white collar workers than for any other type. The average per capita capital expenditure for white collar workers in our economy is $3,000. The corresponding figure for blue collar workers is $25,000 and for

farm workers is $35,000. Productivity is harder to measure in clerical areas. Perhaps part of the problem is that managers feel that office workers are not really producing anything anyway, so why spend any money on them. This attitude is crumbling in the face of office automation, but it will be a while before it disappears.

4.2 ELEMENTS OF A PROGRAMMING ENVIRONMENT

When we talk about a programming environment, we are talking about a whole complex of support facilities for a programmer. Specifically, the points covered in the idea include:

1. Tools support - access to convenient, up to date technologies.
   a) Interactive systems.
   b) Flexible data editing facilities.
   c) Text formatting and document preparation.
   d) Electronic mail.
   e) Interactive compilers.
   f) Flexible interactive terminal systems.
2. Architectural environment.
   a) Comfortable workspaces.
   b) Adequate storage for documentation and listings.
   c) Library facilities.
   d) Conference and meeting facilities.
3. Team support.

a) Data entry personnel.

b) Program and data libararians.

c) Technical writers.

d) Management aides.

e) Adequate secretarial support.

4. Social support.

a) Professional development seminars.

b) Training opportunities.

c) Access to technology.

The requirements of a programmer in a systems development environment are basically threefold. They are:

1. Privacy for program writing activities.
2. Meeting areas for social interaction on project work.
3. System access for program testing.

Anything that undermines these requirements will undermine the programming environment generally.

4.3 IMPROVEMENTS IN PRODUCTIVITY DUE TO IMPROVED ENVIRONMENT

From FMSO's point of view, the most interesting question is how much productivity will be improved by improving the development environment. This will be necessary for any cost justification of improvement. It turns out that this will not be an easy question to answer for a variety of reasons. First of all, there is no real system of productivity measurement in place at FMSO, so we have no way to measure productivity. Secondly, FMSO is a unique environment. In

most ways, it is far behind current computer technology. It is doing development in a computer environment that most organizations scrapped ten years ago. Its management climate is more up to date. The improved programming technologies are given in the development handbooks, the staff is aware of them and seems dedicated to implementing those that can be adapted to FMSO's situation.

It is this combination of factors that makes comparison with industrial experience difficult. Most of the research on productivity improvement deals with the improved programming technologies. There is little work done on a comparison between interactive vs. batch modes of program development. The reason for this is simple. Everybody considers interactive coding to be such an improvement that nobody has bothered to research the question lately. There were a few tentative papers on the subject in the late 1960's, but there has been little recently*.

The programming styles of the workers will be different in interactive environments than they will in a batch environment. When a programmer does development work in a batch mode, he has to keep several projects going concurrently.

* See Harold Sackman's article, "Exploratory Experimental Studies Comparing Online and Offline Programming Performance" published in the Communications of the Association for Computing Machinery in January 1968. Sackman's article concluded that there was about a 20% improvement in programmer time using an interactive system. It should be noted that study used the relatively crude interactive systems of the late 1960's. Today's screen oriented systems are orders of magnitude better than the early interactive systems.

This way, he has something to do while waiting for the output of his last computer run. This also means that there is a setup cost each time he shifts gears from one project to another. In an interactive environment, a programmer is able to concentrate on a single project until it is completed. He does not have to plan his work around the difficulties of computer access. This would suggest that interactive systems development should have high payoff in a maintenance oriented environment like FMSO's. For most maintenance work, the changes to a program tend to be small and could be completed quickly. An interactive system could also be useful in controlling the forms and documentation associated with program development.

4.4 CONCLUSIONS

More attention needs to be given to the issue of programming environment at FMSO. The major points that need improvement are the physical and technical environment. These have been amply covered elsewhere. But simply improving these aspects of the environment is not enough. When the new system is installed, work should also begin on providing support to developers in the form of improved tools, secretarial assistance and general programming team support. This will allow FMSO to reap full benefit from the new technologies.

Chapter 5

A PRODUCTIVITY MEASUREMENT SYSTEM

5.1 INTRODUCTION

This chapter will discuss the measurement of productivity within the software development and maintenance process. Productivity measures, when defined as a measure of output divided by a measure of input, are used for measuring the efficiency of any production process. Productivity measures can provide information on efficiency at all levels of an organization. These levels include various projects and departments, as well as the entire organization. This chapter will examine the purposes of productivity measurement, the productivity measurement problem in general, various measures of programming productivity and a brief discussion of the implementation of a productivity measurement system.

5.2 PURPOSES OF PRODUCTIVITY MEASUREMENT

The primary purpose for measuring productivity in the software development and maintenance process is to provide information for use in the three major phases of software management. These phases are the planning, control and evaluation of the entire software process as well as individual projects within the process.

5.2.1 Planning

The primary function of the planning phase of software management is to provide a prior estimate of resources required for supporting the software development and maintenance process, either on a project basis or on a recurring basis for the entire software department. The planning phase allows for the establishment of budgets for the department and projects as well as subbudgets for each of the input factors, especially labor, which are required in the software process. The productivity measure is used, along with an estimate of the amount of output required for the project or department, to generate an estimate of the amount of input(s) required for a given time period. It should be noted that there exists a direct link during this phase between the productivity measure and project cost estimating methods. On a project basis, the productivity measurement problem is equivalent to the project cost estimation problem.

5.2.2 Control

The control phase of software management entails the determination of the extent of progress of the software process and may be applied at both the department and project level. Progress may be measured on two dimensions, budget and schedule. On the budget dimension, actual expenditures are compared with planned expenditures and variances are ex-

amined. Adequacy of progress is measured by these variances. The planned expenditures, which were generated during the planning phase, usually serve as the standards for measuring progress. However, the plans are subject to modification due to unforeseen contingencies.

On the schedule dimension, actual elapsed times are compared with planned elapsed times and, again, variances are examined. As in the case of budgets, planned schedules serve as standards except when modified by unexpected events. Productivity measures are used in this phase to assist in the determination of actual budgets and schedules, as well as to assist in the modification of plans when contingencies occur.

5.2.3 Evaluation

The evaluation phase of software management concerns itself with the determination of how well the software process is meeting the goals of the organization. This determination may be made at the department level, the project level or any intermediate level. An integral part of this determination of the adequacy of the entire process is an evaluation of the efficiency of the process. Productivity measures, functioning as pure efficiency measures, are useful during the evaluation phase.

5.3 PRODUCTIVITY MEASUREMENT IN GENERAL

Productivity is defined as the relationship between the output of goods and services and the inputs used to produce those outputs. Two general types of productivity ratios are generally used: total factor productivity and partial factor productivity ratios. Total factor ratios include all of the inputs in the production process, while partial factor ratios do not. The inputs or factors of production are generally classified into three major categories: labor, capital and materials. A total factor ratio may be able to distinguish subclasses within each of these major categories. For example, labor is not homogeneous and different types of labor, such as skill levels, may be appropriate. Any factor may be used but labor is in common usage as a partial measure with the input measure generally being man-hours or man-years. Note that the use of partial measures may be misleading since changes in one input have effects upon all other inputs as well as output. Consequently, an increase in output per man-year indicated by a partial ratio should not be interpreted to mean that the increase is due solely to the increased efficiency of labor. This is because the increase in output is a result of all of the factors of production working together.

Productivity ratios are affected by both short and long-run elements. Probably the most important of the short-run elements is the change in utilization of productive capaci-

-y. Productivity ratios generally vary inversely with changes in the degree of capacity utilization since the fixed inputs cannot be varied with changes in output. Other elements causing short-run changes in productivity are both the level of effort and the learning process which occur as individuals adapt to new methods and equipment. Among those elements causing long-run changes in productivity are changes in the quality of inputs. Such changes are referred to as input-augmenting technological change. Perhaps most important of the long-run elements are changes in the methods of organizing production. These changes in the underlying production function are a result of such items as changes in the organizational structure or changes in managerial ability.

There are several problems involved in the use of productivity measures. These center around the measurement of inputs and outputs and their abilities to measure efficiency. When measuring inputs for use in a productivity measure, it is desirable to ensure that only the inputs that are actually utilized in the production process are used in the measure. This is especially important for the labor input and implies, for example, that only time worked should be utilized in the measure instead of time paid. Although time paid may be of interest since it corresponds to the total cost incurred, it is important that such items as administrative time, etc. be separated out. This will enable man-

agers to focus more carefully upon the separate activities of productive work and other work. Also, it is imperative that only inputs which are homogeneous be aggregated. Using labor as an example again, different skill levels, such as a keypunch operator versus a systems programmer, perform entirely different tasks and should be aggregated only when they are occur within a particular department. It is also preferable to measure inputs in terms of physical units. Value units may also be utilized but physical units should be used if available.

A primary problem with the measurement of outputs is that convenient measures are sometimes not available either in physical units or value units. This output measurement problem occurs principally within public sector organizations. In this case, there is usually no accepted operational definition of what the outputs really are (national defense, welfare, etc.), and the outputs are not traded in any markets so that value measures are unavailable, also. Consequently, most of the output measures in use are actually intermediate outputs or, simply, inputs to further processes. Such measures are weak, at best.

In the cases where inputs and outputs may not be precisely measured, the productivity measure becomes susceptible to perverse incentives and gaming. This implies that the control and evaluation phases of management may focus upon faulty indicators. For example, if the output measure is

not truly output, there is some danger that generation of input may be enccuraged rather than output. This increase in inputs does not necessarily imply increased output. Also, use of such measures may provoke the generation of useless output. In one instance in the public sector, the measure was square feet of buildings cleaned. This resulted in many areas being cleaned twice daily and some areas not at all.

Another major problem with productivity measures is how to deal with the quality of output. Ostensibly, the quality of output should be held constant in productivity measures but changes in quality may be difficult to measure. Quality changes can cause difficulties in both directions; quality deterioration may cause the measure to increase, while quality improvement may cause the measure to decrease.

A final problem is that changes in one partial productivity measure can be misleading concerning its effects upon the entire production process. There are numerous ways to obtain a given output with several inputs. Technical efficiency exists when, at a constant output level, reduction of one input necessitates an increase in another input. However, economic efficiency exists when the relative marginal costs of utilizing all inputs in production of the output are the same. Technical efficiency is required for economic efficiency but not vice versa. Note that neither of these two concepts of efficiency are captured by partial produc-

tivity measures. Overdependence upon partial measures for control and evaluation can result in underutilization of particular inputs which leads to less rather than more efficient use of resources.

5.4 MEASURING PROGRAMMING PRODUCTIVITY

5.4.1 Introduction

Keeping in mind the above discussion of productivity measures in general, we can now begin to discuss the software programming productivity problem. Programming is one of the major inputs into the software development and maintenance process. Note, however, that programming is an input to this process; it is not the output. The output of the software process is usable software. Other definitions and measures of this output abound but all are simply further derivations on this one concept. Most output measures which are currently being used in programming productivity suffer from being either pure or intermediate inputs. Physical measures, such as programs, do not address the problem that users of software are not interested in programs but are interested only in the output from the programs. Value measures, such as revenues and sales, are available for private sector entities but are not available for public sector organizations, such as FMSO. Because the definition of the output from the software process is so equivocal, several alternative measures are being utilized currently. These

generally cluster around lines of code produced in the programming process, functions which the program performs and functions which the user performs when utilizing the program. Additionally, most variations on these measures use some measure of labor to generate a (partial) productivity measure. Productivity measures using these three major types of output measures as well as an additional one, completed projects, are evaluated below.

5.4.2 Lines of Code

The productivity measure using lines of code is usually lines of code per labor unit, where the labor unit may be man-days, man-weeks, man-months, etc. Lines of code is a physical measure; however, it measures an input into the software development and maintenance process not an output. Lines of code are necessary to produce a software program but cannot measure how the program functions.

A major difficulty in implementing the use of lines of code as a productivity measure is to define exactly what a line consists of. Programs consist of more than executable lines of code. In addition to the executable statements, there may be job control language, comment statements, data declarations and macro-instructions. Depending upon what is or is not counted as a line, various measures of lines of code may differ by factors of two or more.

Another major difficulty in using lines of code as a measure concerns its poor capabilities when measuring non-coding tasks. The entire program development process requires much more than coding. Most lines-of-code measures attempt to deal with this by using long-run measures which have some average amount of noncoding work built into them. However, the application of lines of code per programmer-month to such tasks may result in questionable results in specific circumstances.

These measures also tend to penalize higher level languages. The initial portions of the software development cycle, such as the determination of user requirements, specifications and test cases, as well as later portions, such as the writing of documentation, do not depend upon the language utilized. Since higher level languages tend to require fewer source statements to program than lower level languages, combination of the coding portion with the language-independent portions of development results in an apparent lesser productivity when using higher level languages. This apparent paradox exists because the productivity measure is just that and is not a measure of total cost.

A problem related to the higher level language problem is that lines of code does not adequately deal with quality differentials in different programs. Some efforts have been made to permit the introduction of quality measures within lines of code via the use of complexity metrics. Because

the field of complexity metrics is still undergoing development with no dominant metric available, the use of such metrics has not yet offered a precise solution to the quality measurement problem.

The last major problem with lines of code is that it perpetuates the myth that coding is the predominant activity in software development and maintenance. This may have been the case in the early days of programming, but in, for example, modular programming there may be no new code created during a particular project.

A relatively minor problem appears to be that, when such measures are applied to subtasks in a project and then aggregated, the aggregation of the subtask measures to a single overall measure is performed incorrectly. The correct method of aggregation depends upon whether the subtasks are performed simultaneously or sequentially. If they are performed simultaneously, the aggregate measure will be larger than any of the subtask measures, and, if they are performed sequentially, the aggregate measure will be smaller than any of the subtask measures. Combinations require that sequential subtask measures be aggregated first, followed by aggregation of the remaining simultaneous measures.

As with all productivity measures, lines of code is susceptible to gaming. Programmers may be able to generate apparent increases in productivity where none really exists as well as apparently prevent decreases in productivity where such a decrease has actually occurred.

Despite the problems given above, lines of code does have a major advantage in that it is relatively easy to measure. With judicious use, lines of code may offer an excellent method for measuring programmer productivity. The following statement illustrates this:

> There is still a great deal to be learned about quality and productivity normalized against lines of code. We have not explored the limits of knowledge, and comparisons between different kinds of programs--with lines of code counted the same way for both--almost daily yield new insights and discoveries. It is premature to abandon this method, just when results are becoming encouraging.[5]

5.4.3 Program Functions

A productivity measure has been proposed and tested which is based upon functions performed by the program. The specific measure is labor units (specifically, man-hours) per function. Note that this is not actually a productivity measure, but is the inverse of a productivity measure. As in the case of lines of code, program functions measures an input into the software development and maintenance process instead of an output. Although it is able to measure how a program functions, this measure does not capture how users evaluate or utilize a particular piece of software.

---

[5] This quote (page 51) and some of the above discussion is taken from Jones, T.C., "Measuring Programming Quality and Productivity," IBM Systems Journal, Vol. 17, No. 1, 1978, pp. 39-63.

Perhaps the most difficult aspect of using program functions as a measure is the definition of a program function. Prior applications have been limited to structured programming environments only. Within this structured approach a function is defined to be a paragraph. Depending upon the particular language, a paragraph and, therefore, a function may be a procedure or a (sub)routine. This also corresponds to the concept of a module. Given a specific method of measuring program functions, this measure is relatively easy to calculate. However, this measure is also susceptible to gaming, depending upon the extent to which the individual programmer can control the structure of the program.[6]

5.4.4 User Functions

A productivity measure has been proposed which is based upon external attributes or functions which are activated by the user. The general approach in this case is to determine the external or user-oriented manifestations of any application software. In practice, this is accomplished by counting the number of external user inputs, outputs, inquiries and master files delivered by the project.

---

[6] A discussion of program functions may be found in Crossman, T.D., "Taking the Measure of Programmer Productivity," Datamation, May 1979, pp. 144-147.

The counts of each of these four factors may be weighted to attempt to reflect the relative value of each to the user. Albrecht suggests specific weights which were found to be useful in one particular organization. Additionally, the weighted sum may be adjusted to account for extraordinary circumstances. The result is a measure of function counts for a specific project. The actual measure utilized by Albrecht was hours worked per function count, which is the inverse of a productivity measure.

The measure of user functions per labor unit offers the considerable advantage of actually attempting to measure the output, user functions, of the software process. In this respect, it corresponds more closely to a true productivity measure than any of the other programming measures discussed above. Since it is more output-oriented, it is much more difficult to game than any of the other measures.

The actual measurement of user functions in specific circumstances may be nontrivial, however. For example, it is conceivable that one may have a difficult time discerning whether a particular function is an input or an inquiry. If a different weight is allowed for inputs than is allowed for inquiries, the selection of particular user functions may have an unintended effect upon the actual value generated by the measurement procedure.[7]

---

[7] User functions are discussed in Albrect, A.J., "Measuring Application Development Productivity," Proceedings of the SHARE/GUIDE/IBM Application Development Symposium, October 1979, pp. 83-92.

5.4.5 Completed Projects

Another possible measure is completed projects per labor unit. This measure offers the advantage of being exceptionally easy to implement and use. However, unless a reasonably large number of different categories of projects are maintained, this measure is also exceptionally susceptible to gaming. This is especially true if the individual to be measured has any input into the selection of which projects are to be programmed by whom. Also, the existence of a large number of categories compromises the ease-of-use advantage of this measure. Similarly, it presents the additional problem of cross comparisons of different measures when looking at a single programmer.

5.4.6 Summary

Each of the above measures has unique advantages and disadvantages. This ensures that there is no one dominant measure for all situations. Since FMSO is both a development and a maintenance activity, these measures must be examined for their specific comparative advantages in the development and/or maintenance processes. Along this dimension, the two functional measures, program functions and user functions, are useful primarily in the development process. The other two measures, lines of code and completed projects, may be used for both development and maintenance. The measures also have relative advantages and dis-

advantages when used in either applications or systems programming projects. Note, additionally, that there are difficulties in making comparisons across different languages or organizations but here we are interested only in measuring the productivity of the programming process within FMSO.

Hence, it is suggested that a pilot project be set up to collect data on a number of software projects and that several measures be evaluated using these data. This would result in two or possibly three measures being selected for further testing on a much larger sample. After a reasonable number of projects have been examined, then the measures may be further evaluated to produce an operational productivity measurement system.

5.5 IMPLEMENTATION OF A PRODUCTIVITY MEASUREMENT SYSTEM

Productivity measurement systems, despite the advantages discussed above, are not used universally. Perhaps the major reason for this is because these systems are not costless. Resources are required, from both managers and programmers, in order to implement such systems. All of the measurement systems above are based on daily inputs by individual programmers in order to keep track of labor units. Also, analysis of the project is necessary in order to generate alternative output measures. However, such systems are, in general, cost-effective based upon their general us-

age. In addition to the measurement system's contribution to the potential increase in productivity, there are several other reasons that FMSO should begin to implement a productivity measurement system.

The major reason is that a similar system, designed to associate specific resources with specific software projects, will be implemented in the near future by DOD. The Air Force is the lead service in the testing of the Software Acquisition Resource Expenditure (SARE) reporting system. New directives will require such reports for all software developed by and for DOD. Consequently, the forthcoming SARE reporting system will be much easier to implement if a data collection system has been tested and is being utilized by FMSO.

Another possible reason for movement to a productivity measurement system by FMSO is to provide the basis for quantitative justification of resources required for particular projects. Under the commercial activities program, all non-mission-essential activities are subject to private sector provision. In the case of software development and maintenance, this program would require FMSO to bid on particular projects along with private sector software houses. Such bids must be auditable, which implies that productivity information must be quantitatively-based and verifiable. A related aspect is that the NARDAC's are moving to a NIF funding basis instead of a mission-funded basis. It is not

impossible that FMSO and its customers could be moved to such a fund accounting basis in the future.

Although a productivity measurement system can provide reasonably precise and accurate information, it cannot increase productivity on its own. Other chapters of this report provide other techniques and methods for dealing with productivity enhancement at FMSO. As the following remark indicates, these other factors are also important.

> How workers feel about their jobs, about their fellow workers, about management, and about the organization, may be more important in influencing productivity than is the particular way they are instructed to do their work, the formal organizational structure, or even financial incentives.[8]

---

[8] This is found on page 1038 of Nelson, R.R., "Research on Productivity Growth and Productivity Differences: Dead Ends and New Departures," The Journal of Economic Literature, Vol. 19, No. 3, September 1981, pp. 1029-1064.

Chapter 6

A PROJECT PLANNING SYSTEM

6.1 INTRODUCTION

Software project management has never been an easy task. The area is characterized by slipped schedules, overdue deliveries and systems that do not work as they were intended. Over the years, we have gotten smarter about software system development. We know more about how to do it, and we settle for less than our most optimistic hopes. A manager thrown into this environment for the first time quickly comes to appreciate Fred Brooks metaphorical use of the LaBrea Tar Pits to characterize software development projects[9].

A great deal has been written about software project management techniques lately. Naturally, most of the writeups are of success stories. In the normal manner of success stories, they make the project management process seem easier than it probably was. This is where the "Grass is Greener" syndrome begins to come in. We know that in our own organizations, there are problems that sometimes get out of hand. These success stories sometimes make us think that everybody else has things under control. The answer then seems to be to take the methods that have (presumably)

---

[9] Fred Brooks, The Mythical Man-month, p. 3.

worked well elsewhere and adapt them to our organizations.

This would be a great idea if it were possible, but unfortunately, there are a few hitches in the process. Software projects are not simple, and they are not standardized across organizations. The standards for measuring productivity in different organizations are going to be vastly different. We can learn a lot about installing a project planning and control system by observing the techniques that have worked elsewhere, but we have to be careful. It is especially dangerous to take productivity figures from one organization as necessarily being indicative of what we can expect. First of all, we have to know what they think productivity is and how they account for it. That information rarely appears in the articles in sufficient detail to allow us to reconstruct the measures used by the original researchers, much less use them in our own organizations.

What we will have to do is to develop our own models (probably patterned on those developed elsewhere), gather data on how they work, and gradually develop our own system for project estimating. This implies that a project estimation model has to be tailor made for a given organization.

6.2 PROJECT PLANNING AND CONTROL SYSTEMS

Just because project planning and control systems must be tailored to a given organization, this does not mean that there is not considerable guidance available on successful approaches to project planning systems. There are two major approaches we would recommend considering as models for a project planning system at FMSO. The first of these is the SLIM system developed by Lawrence Putnam[10]. Putnam's model is based on the Rayleigh Curve and can be used to estimate effort required and timing of effort for medium to large scale software projects. The SLIM model is available over time sharing services and could serve as a useful first approach to developing local project planning models at FMSO. A good source for information on the SLIM model and the way in which it is used for project planning is given by Vorgang[11]. The SLIM model is basically a macro approach to cost and effort estimation that makes reasonable assumptions about the type of work being done. It seems to offer a little less flexibility than you would get by developing your own model, but it is probably a good place to start investigating the subject.

---

[10] A detailed account of Putnam's model is found in Lawrence Putnam, "A General Empirical Solution to the Macro Software Sizing and Estimating Problem", IEEE Transactions on Software Engineering, July 1978, pp. 345 - 361.

[11] Blair Roland Vorgang, "A Macro Approach to Software Resource Estimation and Life Cycle Control", Master's Thesis in Computer Systems Management, Naval Postgraduate School, December 1981.

The next project planning model is the COCOMO (for COnstructive COst MOdel) system developed by Barry Boehm[12] of TRW. Boehm provides a detailed description of the COCOMO system in his book. The data used to derive the model comes from experience in software development at TRW. Unfortunately, this experience may not be entirely relevant to FMSO's software development situation. There is not a sufficient number of COBOL data points used to derive the parameters of the model. The COCOMO model probably works better in the scientific computing environment common in the aerospace industry than it does in a purely data processing environment.

However, Boehm is detailed enough about how the models are put together that he provides an excellent pattern to follow in developing your own project planning systems. The COCOMO system is divided into basic and intermediate models. Both systems are used to predict the effort required to develop a software product. The basic system uses a single predictor variable, number of lines of source code required, and three different modes of development. The three modes are:

> Organic Mode - In organic mode, relatively small software teams develop software in a highly familiar, in-house environment. Most people connected with the project have extensive experience in working with related systems

---

[12] The COCOMO model as well as a number of other important issues in project planning and cost estimation are described in Barry Boehm, Software Engineering Economics, published by Prentice-Hall, Inc., 1981.

within the organization, and have a thorough understanding of how the system under development will contributed to the organization's objectives.

Embedded Mode - In embedded mode, the distinguishing factor of a project is a need to operate within tight constraints. The product must operate within (is embedded in) a strongly coupled complex of hardware, software, regulations and operational procedures, such as an electronic funds transfer system or an air traffic control system.

Semidetached Mode - The semidetached mode is an intermediate level between the organic and embedded modes. The project contains mixtures of both embedded and organic mode characteristics.

FSMO's mode of project developement could normally be characterized as organic mode, although some of the projects undertaken by the Environmental Group would probably be considered to be embedded mode.

At this basic level, Boehm reports that COCOMO predictions come within a factor of 1.3 of actuals 29% of the time and within 2.0 of actuals 60% of the time. If these results do not seem overly impressive you can move to what Boehm calls intermediate COCOMO. In this enhancement of the model, Boehm adds an additional fifteen factors or "cost driver attributes" that are given below. The names of the variables are those used in Boehm's model itself.

Product Attributes

RELY - required software reliability

DATA - data base size

CPLX - product complexity

Computer Attributes

TIME - execution time constraint

STOR - main storage constraint

VIRT - virtual machine volatility

TURN - computer turnaround time

Personnel Attributes

ACAP - analyst capability

AEXP - applications experience

PCAP - programmer capability

VEXP - virtual machine experience

LEXP - programming language experience

Project Attributes

MODP - modern programming practices

TOOL - use of software tools

SCED - required development schedule

Each of these parameters are assigned weights (called "effort multipliers" by Boehm), and these weights are used multiplicativly to adjust parameters in the model. Boehm claims that with intermediate COCOMO, he is within 20% of actuals 68% of the time.

In many ways, the type of data collected by Boehm for intermediate COCOMO is similar to that already considered by FMSO for its own project estimation data gathering. This is not surprising since Boehm's model passes what could be called a "reasonable man" test. The parameters in the model are those that an experienced professional would probably expect to find contributing to effort required in a software development project. With Boehm's model, or any software development model, one must be careful how it is applied. The development of such an estimation model is an evolutionary process and there are many opportunities for problems. Putnam cautions that "I have found that manpower data accumulated to a yearly value is not more accurate than +/- 10-15% of the reported value. If the data are examined at shorter intervals, the percentage variation tends to be even greater"[13]. It will take time to figure out what the bias caused by accounting problems is in your organization and how these should be accounted for in the model. In addition, there is probably a "Hawthorne Effect" present in software management. As a model is developed, the tighter controls are likely to bring about increased programmer awareness of management goals. To a certain extent, the effort projections may become self fulfilling prophecies. They determine the time alloted to the software project, and

---

[13] Lawrence Putnam, "A General Empirical Solution to the Macro Software Sizing and Estimating Problem", IEEE Transactions on Software Engineering, July 1978, p 346.

at the end of that time, the project is declared done unless it has truly major deficiencies.

Whatever model is used for projections in the FMSO environment, it should be kept in mind that software effort estimation is not an exact science by any means. The models will have to be developed in an evolutionary fashion and will have to be geared to FMSO's unique situation. Moreover, as technology changes (assuming, for example, that the recommendation to acquire a development tools machine is taken), then it is to be expected that it will be necessary to change the models used as well. It will probably take a number of years to come up with a useful model.

6.3 CONCLUSIONS

FMSO should begin some preliminary work on developing a project planning and effort estimation model. Some of the work has already been done in that there has been data gathering done on the time required for projects, and the forms used are similar to those in use elsewhere. The next step will be to analyze the data collected, to begin to develop some computer models of effort and then attempt to validate these models on on-going FMSO software development projects. This work need not wait until the acquisition of a development tools system. It could probably be initiated using any of the micros currently in place at FMSO.

REFERENCES

1. Albrecht, A. J., "Measuring Application Development Productivity", in SHARE-GUIDE, 1979, pp. 83 - 92.

2. Baker, F. T., "Chief Programmer Team", IBM System Journal, v. 11, no. 1, 1972.

3. Baker, F. T., "System quality through structured programming", Proceedings of the 1972 FJCC, v. 41, pt. 1, AFIPS Press, Montvale, NJ, 1972.

4. Bernstein, M. I., "Hardware is Easy: It's Software That's Hard", Datamation, v. 24, no. 12, November 15, 1978, pp. 32-36.

5. Boehm, Barry, "An Experiment in Small-Scale Application Software Engineering", IEEE Transactions on Software Engineering, v. SE-7, no. 5, September 1981, pp. 482-493.

6. Boehm, Barry, Software Engineering Economics, Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, 1981.

7. Brooks, Fredrick P., The Mythical Man-Month, Addison-Wesley Publishing Co., Inc., Reading, Mass., 1975.

8. Conroy, James, Fuqua, Michael and Sisco, James, "A Survey of Software Quality ASsurance Methods and an Evaluation of Software Quality Assurance at Fleet Material Support Office", Master's Thesis in Computer Systems Management submitted to the Naval Postgraduate School, Department of Administrative Sciences, December 1982.

9. Daly, Edmund B., "Organizing for Successful Software Development", Datamation, v. 25, no. 14, December 1979, pp. 106-120.

10. Green, James F. and Selby, Brenda F., Dynamic Planning and Control of Software Maintenance: A Fiscal Approach, M.S. Thesis submitted to the Naval Postgraduate School, Department of Administrative Sciences, December 1981.

11. Habermann, A. N., "Tools for Software System Construction", Software Development Tools, May 1979, pp. 10 - 21, also pp. 363 - 371 in Jones, Programming Productivity: Issues for the Eighties.

12. Herbert, Leo, Auditing the Performance of Management, Lifetime Learning Publications, Belmont, CA, 1979.

13. Howden, William E., "Life-Cycle Software Validation", Computer, v. 15, no. 2, February 1982, pp. 71-78.

14. Hughes, Gary J., "A Study of Programmer Productivity Metrics for Fleet Material Support Office (FMSO)", M.S. Thesis submitted to the Naval Postgraduate School, Department of Administrative Sciences, June 1983.

15. Jensen, Randall W. and Tonies, Charles C., Software Engineering, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1979.

16. Jones, Capers, "Program Quality and Programmer Productivity", IBM Technical Report TR 02.764, pp 1, 42 - 78, also pp. 124 - 161 in Jones, Programming Productivity: Issues for the Eighties.

17. Jones, Capers, "Measuring Programming Quality and Productivity", IBM Systems Journal, Vol. 17, No. 1, 1978, pp. 39 - 63, also pp. 9 - 33 in Jones, Programming Productivity: Issues for the Eighties.

18. Jones, T. C., "The Limits of Programming Productivity", Proceedings of the Joint SHARE/GUIDE/IBM Applications Development Symposium, October 1979, pp. 77 - 82, also pp. 401 - 410 in Jones, Programming Productivity: Issues for the Eighties.

19. Jones, Capers (ed.), Programming Productivity: Issues for the Eighties, IEEE Catalog No. EHO 186-7, IEEE Computer Society, P.O. Box 80452, Los Angeles, California, 1981.

20. Katzan, H., Systems Design and Documentation: An Introduction to the HIPO Method Van Nostrand Reinhold, New York, 1976.

21. Kendall, R. C. and Lamb, E. C., "Management Perspectives on Programms Programming and Productivity", presented at GUIDE 45, Atlanta, Georgia, November 1977, also reprinted in Programming Productivity: Issues for the Eighties, IEEE Catalog No. EHO 186-7, IEEE Computer Society, P.O. Box 80452, Los Angeles, California, 1981.

22. Mair, William C., Wood, Donald R., and Davis, Keagle W., Computer Control and Audit, The Institute of Internal Auditors, 249 Maitland Ave., Altamonte Springs, Florida, 1978.

23. McCracken, Daniel D., "The Changing Face of Applications Programming", Datamation, v. 24, no. 12, November 15, 1978, pp. 24-30.

24. McCue, G. M., "IBM's Santa Teresa Laboratory - Architectural Design for Program Development", IBM Systems Journal, Vol. 17, No. 1, 1978, pp. 4 - 25 also pp. 312 - 333 in Jones, Programming Productivity: Issues for the Eighties.

25. Metzger, Phillip, Managing a Programming Project, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1973.

26. Myers, G. J., Software Reliability, John Wiley and Sons, Inc., New York, 1976.

27. Meyers, G. J., The Art of Software Testing, John Wiley and Sons, Inc., New York, 1979.

28. Patrick, Robert L., "The Productivity Gap", Datamation, v. 25, no. 14, December 1979, pp. 131-132.

29. Peercy, D. E., "A Software Maintainability Evaluation Methodology", IEEE Transactions on Software Engineering, v. SE-7, no. 4, July 1981, pp. 343-352.

30. Petersen, Perry, "Project Control Systems", Datamation, v. 25, no. 7, June 1979, pp. 147-162.

31. Putnam, Lawrence, "A General Empirical Solution to the Macro Software Sizing and Estimating Problem", IEEE Transactions on Software Engineering, Vol. SE-4, No. 4, July 1978, pp. 345 - 361.

32. Putnam, Lawrence and Fitzsimmons, E., "Estimating Software Costs", Datamation, September 1979, pp. 189 - 198, October 1979, pp. 171 - 178, November 1979, pp. 137 - 140.

33. Reifer, D. J., and Trattner, S., "A Glossary of Software Tools and Techniques", Computer, July 1977, .pp 52 - 60, also pp. 344 - 352 in Jones, Programming Productivity: Issues for the Eighties.

34. Sackman, Harold, Erikson, W. J. and Grant, E. E. "Exploratory Experimental Studies Comparing Online and Offline Programming Performance", Communications of the ACM, v. 11, no. 1, January 1968, pp. 3-10.

35. Shaw, John C., and Atkins, William, Managing Computer System Projects, McGraw-Hill, New York, 1970.

36. Shneiderman, Ben, Software Psychology, Winthrop Publishers, Inc., Cambridge, Mass., 1980.

37. Spooner, Daniel J., "A Study of Quantitative Measurements of Programmer Productivity for Fleet Material Support Office (FMSO)", Master's Thesis in Computer Systems Management at the Naval Postgraduate School, December 1982.

38. Thayer, R. H., Pyster, A. B., and Wood, R. C., "Major Issues in Software Engineering Project Management", IEEE Transactions on Software Engineering, v. SE-7, no 4, July 1981, pp. 333-342.

39. Vorgang, Blair Roland, "A Macro Approach to Software Resource Estimation and Life Cycle Control", Master's Thesis in Computer Systems Management, Naval Postgraduate School, December 1981.

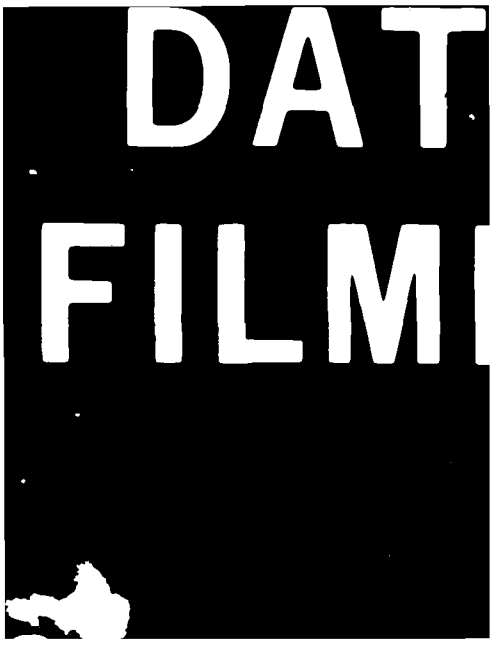
40. Walston, C. E. and Felix, C. P., "A method of programming measurement and estimation", IBM Systems Journal, Vol. 10, No. 1, pp. 10 - 29, also reprinted in Programming Productivity: Issues for the Eighties, IEEE Catalog No. EHO 186-7, IEEE Computer Society, P.O. Box 80452, Los Angeles, California, 1981.

41. Wolverton, R. W., "The Cost of Developing Large-Scale Software" reprinted in Programming Productivity: Issues for the Eighties, IEEE Catalog No. EHO 186-7, IEEE Computer Society, P.O. Box 80452, Los Angeles, California, 1981.

42. Yourdon, Edward, Managing the Structured Techniques, Second Edition, Prentice-Hall, Englewood Cliffs, New Jersey, 1979.

INITIAL DISTRIBUTION LIST

	No. of Copies
Dean of Research Code 014 Naval Postgraduate School Monterey, CA 93943	1
Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Library, Code 1424 Naval Postgraduate School Monterey, CA 93943	2
Professor Richard S. Elster Chairman, Dept of Administrative Sciences Naval Postgraduate School Monterey, CA 93943	1
Professor Dan C. Boger Code 54Bk Naval Postgraduate School Monterey, CA 93943	10
Professor Norman R. Lyons Code 54Lb Naval Postgraduate School Monterey, CA 93943	10
CAPT Francis Filipiak, USN Code 9 Fleet Material Support Office (FMSO) Mechanicsburg, PA 17055	5


DAT
FILM